(51) International Patent Classification: G06F 9/38

(21) International Application Number: PCT/US00/12499
(22) International Filing Date: 5 May 2000 (05.05.00)
(30) Priority Data: 09/313,229 17 May 1999 (17.05.99) US

(71) Applicant: SUN MICROSYSTEMS, INC. [US/US]; 901 San Antonio Road, Palo Alto, CA 94303 (US).


(74) Agent: PARK, Richard; Suite 201, 508 Second St., Davis, CA 95616 (US).


Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: SUPPORTING SPECULATIVE EXECUTION BY SELECTIVELY VERSIONING MEMORY UPDATES

(57) Abstract

One embodiment of the present invention provides a system that facilitates speculative execution through selective versioning of memory elements located in a system heap. The system includes a primary thread that executes program instructions and a speculative thread that speculatively executes program instructions in advance of the primary thread. The primary thread generally accesses a primary version of a memory element, and the speculative thread generally accesses a corresponding speculative version of the memory element. During a write operation by the primary thread, the system performs the write operation to the primary version of the memory element. It also checks status information associated with the memory element to determine if the memory element has been read by the speculative thread. If so, the system causes the speculative thread to roll back so that the speculative thread can read a result of the write operation. If not, the system performs the write operation to the speculative version of the memory element if such speculative version exists. In a variation on the above embodiment, during a read operation by the speculative thread, the system determines if the speculative version of the memory element exists. If so, the system reads the speculative version of the memory element. If not, the system reads the primary version of the memory element. The system next updates status information associated with the memory element to indicate that the memory element has been read by the speculative thread.
FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>Albania</td>
<td>ES</td>
<td>Spain</td>
<td>LS</td>
<td>Lesotho</td>
<td>SI</td>
<td>Slovenia</td>
</tr>
<tr>
<td>AM</td>
<td>Armenia</td>
<td>FI</td>
<td>Finland</td>
<td>LT</td>
<td>Lithuania</td>
<td>SK</td>
<td>Slovakia</td>
</tr>
<tr>
<td>AT</td>
<td>Austria</td>
<td>FR</td>
<td>France</td>
<td>LU</td>
<td>Luxembourg</td>
<td>SN</td>
<td>Senegal</td>
</tr>
<tr>
<td>AU</td>
<td>Australia</td>
<td>GA</td>
<td>Gabon</td>
<td>LV</td>
<td>Latvia</td>
<td>SZ</td>
<td>Swaziland</td>
</tr>
<tr>
<td>AZ</td>
<td>Azerbaijan</td>
<td>GB</td>
<td>United Kingdom</td>
<td>MC</td>
<td>Monaco</td>
<td>TD</td>
<td>Chad</td>
</tr>
<tr>
<td>BA</td>
<td>Bosnia and Herzegovina</td>
<td>GE</td>
<td>Georgia</td>
<td>MD</td>
<td>Republic of Moldova</td>
<td>TG</td>
<td>Togo</td>
</tr>
<tr>
<td>BB</td>
<td>Barbados</td>
<td>GH</td>
<td>Ghana</td>
<td>MG</td>
<td>Madagascar</td>
<td>TJ</td>
<td>Tajikistan</td>
</tr>
<tr>
<td>BE</td>
<td>Belgium</td>
<td>GN</td>
<td>Guinea</td>
<td>MK</td>
<td>The former Yugoslav</td>
<td>TR</td>
<td>Turkmenistan</td>
</tr>
<tr>
<td>BF</td>
<td>Burkina Faso</td>
<td>GR</td>
<td>Greece</td>
<td>ML</td>
<td>Mali</td>
<td>TR</td>
<td>Turkey</td>
</tr>
<tr>
<td>BG</td>
<td>Bulgaria</td>
<td>HU</td>
<td>Hungary</td>
<td>MN</td>
<td>Mongolia</td>
<td>TT</td>
<td>Trinidad and Tobago</td>
</tr>
<tr>
<td>BJ</td>
<td>Benin</td>
<td>IE</td>
<td>Ireland</td>
<td>MR</td>
<td>Mauritania</td>
<td>UA</td>
<td>Ukraine</td>
</tr>
<tr>
<td>BR</td>
<td>Brazil</td>
<td>IL</td>
<td>Israel</td>
<td>MW</td>
<td>Malawi</td>
<td>UG</td>
<td>Uganda</td>
</tr>
<tr>
<td>BY</td>
<td>Belarus</td>
<td>IS</td>
<td>Iceland</td>
<td>MX</td>
<td>Mexico</td>
<td>US</td>
<td>United States of America</td>
</tr>
<tr>
<td>CA</td>
<td>Canada</td>
<td>IT</td>
<td>Italy</td>
<td>NE</td>
<td>Niger</td>
<td>UZ</td>
<td>Uzbekistan</td>
</tr>
<tr>
<td>CF</td>
<td>Central African Republic</td>
<td>JP</td>
<td>Japan</td>
<td>NL</td>
<td>Netherlands</td>
<td>VN</td>
<td>Viet Nam</td>
</tr>
<tr>
<td>CG</td>
<td>Congo</td>
<td>KE</td>
<td>Kenya</td>
<td>NO</td>
<td>Norway</td>
<td>YU</td>
<td>Yugoslavia</td>
</tr>
<tr>
<td>CH</td>
<td>Switzerland</td>
<td>KG</td>
<td>Kyrgyzstan</td>
<td>NZ</td>
<td>New Zealand</td>
<td>ZW</td>
<td>Zimbabwe</td>
</tr>
<tr>
<td>CI</td>
<td>Côte d'Ivoire</td>
<td>KP</td>
<td>Democratic People's Republic of Korea</td>
<td>PL</td>
<td>Poland</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CM</td>
<td>Cameroon</td>
<td>KR</td>
<td>Republic of Korea</td>
<td>PT</td>
<td>Portugal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CN</td>
<td>China</td>
<td>KZ</td>
<td>Kazakhstan</td>
<td>RO</td>
<td>Romania</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CU</td>
<td>Cuba</td>
<td>LC</td>
<td>Saint Lucia</td>
<td>RU</td>
<td>Russian Federation</td>
<td>SD</td>
<td>Sudan</td>
</tr>
<tr>
<td>CZ</td>
<td>Czech Republic</td>
<td>LI</td>
<td>Liechtenstein</td>
<td>SE</td>
<td>Sweden</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DE</td>
<td>Germany</td>
<td>LK</td>
<td>Sri Lanka</td>
<td>SG</td>
<td>Singapore</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DK</td>
<td>Denmark</td>
<td>LR</td>
<td>Liberia</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SUPPORTING SPECULATIVE EXECUTION BY SELECTIVELY VERSIONING MEMORY UPDATES

BACKGROUND

Field of the Invention

The present invention relates techniques for improving computer system performance. More specifically, the present invention relates to a method and apparatus that provides selective versioning of memory objects to support speculative execution of a computer program.

Related Art

As increasing semiconductor integration densities allow more transistors to be integrated onto a microprocessor chip, computer designers are investigating different methods of using these transistors to increase computer system performance. Some recent computer architectures exploit “instruction level parallelism,” in which a single central processing unit (CPU) issues multiple instructions in a single cycle. Given proper compiler support, instruction level parallelism has proven effective at increasing computational performance across a wide range of computational tasks. However, inter-instruction dependencies
generally limit the performance gains realized from using instruction level parallelism to a factor of two or three.

Another method for increasing computational speed is "speculative execution" in which one or more speculative threads execute instructions in advance of a primary thread. If a speculative thread encounters a problem, such as mispredicting the result of a branch operation, the speculative thread performs a "rollback" operation to undo the results of the speculative computation.

In order to undo the results of the speculative computation, updates to memory need to be versioned. The overhead involved in versioning all updates to memory can be prohibitively expensive due to increased memory requirements, decreased cache performance and additional hardware required to perform the versioning.

Fortunately, not all updates to memory need to be versioned; only those updates that form part of the computation's non-speculative state. For example, updates to local variables -- such as a loop counter -- on a system stack are typically only relevant to the speculative thread that is updating the local variables. Hence, versioning updates to these local variables is not necessary.

When executing programs written in conventional programming languages, such as C, it is typically not possible to determine which updates are related to speculative state, and which updates are not. These programs are typically compiled from a high-level language representation into executable code for a specific machine architecture. This compilation process typically removes distinctions between updates to speculative state and non-speculative state.

The same is not true for new platform-independent computer languages, such as the JAVA™ programming language distributed by SUN Microsystems, Inc. of Palo Alto, California. (Sun, the Sun logo, Sun Microsystems, and Java are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.) A program written in the Java programming language
is typically compiled into a class file containing Java byte codes. This class file can be transmitted over a computer network to a distant computer system to be executed on the distant computer system. Java byte codes are said to be “platform-independent,” because they can be executed across a wide range of computing platforms, so long as the computing platforms provide a Java virtual machine.

A Java byte code can be executed on a specific computing platform by using an interpreter or a just in time (JIT) compiler to translate the Java bytecode into machine code for the specific computing platform. Alternatively, a Java byte code can be executed directly on a Java bytecode engine running on the specific computing platform.

Fortunately, a Java bytecode contains more syntactic information than conventional machine code. In particular, the Java bytecodes differentiate between accesses to local variables in the system stack (which tend to contain speculative state) and accesses to the system heap (which tend to contain non-speculative state). Furthermore, programs written in the Java programming language do not allow references through pointers. Such references can make it hard to differentiate accesses to non-speculative state from accesses to speculative state because it is not possible to tell whether a pointer is pointing to the system stack or the system heap.

What is needed is a method and apparatus that uses the syntactic information contained in platform-independent programs, such as Java bytecodes, to reduce the versioning overhead required for speculative execution. More specifically, what is needed is a method and apparatus that supports selective versioning of updates to memory elements located in the system heap.

SUMMARY
One embodiment of the present invention provides a system that facilitates speculative execution through selective versioning of memory elements located in a system heap. The system includes a primary thread that executes program instructions and a speculative thread that speculatively executes program instructions in advance of the primary thread. The primary thread generally accesses a primary version of a memory element, and the speculative thread generally accesses a corresponding speculative version of the memory element. During a write operation by the primary thread, the system performs the write operation to the primary version of the memory element. It also checks status information associated with the memory element to determine if the memory element has been read by the speculative thread. If so, the system causes the speculative thread to roll back so that the speculative thread can read a result of the write operation. If not, the system performs the write operation to the speculative version of the memory element if such speculative version exists.

In a variation on the above embodiment, during a read operation by the speculative thread, the system determines if the speculative version of the memory element exists. If so, the system reads the speculative version of the memory element. If not, the system reads the primary version of the memory element. The system next updates status information associated with the memory element to indicate that the memory element has been read by the speculative thread.

In a variation on the above embodiment, during a write operation by the speculative thread, the system determines if a speculative version of the memory element exists. If not, the system creates the speculative version of the memory element. Next, the system performs the write operation to the speculative version of the memory element.

In a variation on the above embodiment, the system performs a join operation between the primary thread and the speculative thread when the primary thread reaches a point in the program where the speculative thread began
executing. The join operation causes state associated with the speculative thread to be merged with state associated with the primary thread.

**BRIEF DESCRIPTION OF THE FIGURES**

FIG. 1 illustrates a computer system including two central processing units sharing a common data cache in accordance with an embodiment of the present invention.

FIG. 2A illustrates sequential execution of methods by a single thread.

FIG. 2B illustrates speculative execution of a method in accordance with an embodiment of the present invention.

FIG. 3 illustrates the state of the system stack during speculative execution of a method in accordance with an embodiment of the present invention.

FIG. 4 illustrates how memory is partitioned between stack and heap in accordance with an embodiment of the present invention.

FIG. 5 illustrates the structure of a primary version and a speculative version of an object in accordance with an embodiment of the present invention.

FIG. 6 illustrates the structure of a status word for an object in accordance with an embodiment of the present invention.

FIG. 7 is a flow chart illustrating operations involved in performing a write to a memory element by a primary thread in accordance with an embodiment of the present invention.

FIG. 8 is a flow chart illustrating operations involved in performing a read to a memory element by a speculative thread in accordance with an embodiment of the present invention.

FIG. 9 is a flow chart illustrating operations involved in performing a write to a memory element by a speculative thread in accordance with an embodiment of the present invention.
FIG. 10 is a flow chart illustrating operations involved in performing a
join between a primary thread and a speculative thread in accordance with an
embodiment of the present invention.

FIG. 11 is a flow chart illustrating operations involved in performing a
join between a primary thread and a speculative thread in accordance with another
embodiment of the present invention.

DETAILED DESCRIPTION

The following description is presented to enable any person skilled in the
art to make and use the invention, and is provided in the context of a particular
application and its requirements. Various modifications to the disclosed
embodiments will be readily apparent to those skilled in the art, and the general
principles defined herein may be applied to other embodiments and applications
without departing from the spirit and scope of the present invention. Thus, the
present invention is not intended to be limited to the embodiments shown, but is
to be accorded the widest scope consistent with the principles and features
disclosed herein.

The data structures and code described in this detailed description are
typically stored on a computer readable storage medium, which may be any device
or medium that can store code and/or data for use by a computer system. This
includes, but is not limited to, magnetic and optical storage devices such as disk
drives, magnetic tape, CDs (compact discs) and DVDs (digital video discs), and
computer instruction signals embodied in a carrier wave. For example, the carrier
wave may carry information across a communications network, such as the
Internet.
**Computer System**

FIG. 1 illustrates a computer system including two central processing units (CPUs) 102 and 104 sharing a common data cache 106 in accordance with an embodiment of the present invention. In this embodiment, CPUs 102 and 104 and data cache 106 reside on silicon wafer 100. Note that CPUs 102 and 104 may generally be any type of computational devices that allow multiple threads to execute concurrently. In the embodiment illustrated in FIG. 1, CPUs 102 and 104 are very long instruction word (VLIW) CPUs, which support concurrent execution of multiple instructions executing on multiple functional units. VLIW CPUs 102 and 104 include instruction caches 112 and 120, respectively, containing instructions to be executed by VLIW CPUs 102 and 104.

VLIW CPUs 102 and 104 additionally include load buffers 114 and 122 as well as store buffers 116 and 124 for buffering communications with data cache 106. More specifically, VLIW CPU 102 includes load buffer 114 for buffering loads received from data cache 106, and store buffer 116 for buffering stores to data cache 106. Similarly, VLIW CPU 104 includes load buffer 122 for buffering loads received from data cache 106, and store buffer 124 for buffering stores to data cache 106.

VLIW CPUs 102 and 104 are additionally coupled together by direct communication link 128, which facilitates rapid communication between VLIW CPUs 102 and 104. Note that direct communication link 128 allows VLIW CPU 102 to write into communication buffer 126 within VLIW CPU 104. It also allows VLIW CPU 104 to write into communication buffer 118 within VLIW CPU 102.

In the embodiment illustrated in FIG. 1, Data cache 106 is fully dual-ported allowing concurrent read and/or write accesses from VLIW CPUs 102 and 104. This dual porting eliminates cache coherence delays associated with conventional shared memory architectures that rely on coherent caches.
In one embodiment of the preset invention, data cache 106 is a 16K byte 4-way set-associative data cache with 32 byte cache lines.

Data cache 106, instruction caches 112 and instruction cache 120 are coupled through switch 110 to memory controller 111. Memory controller 111 is coupled to dynamic random access memory (DRAM) 108, which is located off chip. Switch 110 may include any type circuitry for switching signal lines. In one embodiment of the present invention, switch 110 is a cross bar switch.

The present invention generally applies to any computer system that supports concurrent execution by multiple threads and is not limited to the illustrated computing system. However, note that data cache 106 supports fast accesses to shared data items. These fast accesses facilitate efficient sharing of status information between VLIW CPUs 102 and 104 to keep track of accesses to versions of memory objects.

**Speculative Execution of Methods**

FIG. 2A illustrates sequential execution of methods in a conventional computer system by a single primary thread 202. In executing a program, primary thread 202 executes a number of methods in sequence, including method A 204, method B 206 and method C 208.

In contrast, FIG. 2B illustrates speculative execution of a method in accordance with an embodiment of the present invention. In FIG. 2B, primary thread 202 first executes method A 204 and then executes method B 206. (For this example, assume that method B 206 returns a void or some other value that is not used by method C 208. Alternatively, if method B 206 uses a value returned by method C 208, assume that method C 208 uses a predicted return value from method B 206.)
As primary thread 202 executes method B 206, speculative thread 203 speculatively executes method C 208. If speculative thread 203 successfully executes method C 208, method C 208 returns and speculative thread 203 is joined with primary thread 202. This join operation involves causing state associated with the speculative thread 203 to be merged with state associated with the primary thread 202.

If speculative thread 203 for some encounters problems in executing method C 208, speculative thread 203 performs a rollback operation. This rollback operation allows speculative thread 203 to reattempt to execute method C 208. Alternatively, primary thread 202 can execute method C 208 non-speculatively and speculative thread 203 can speculatively execute a subsequent method.

There are a number of reasons why speculative thread 203 may encounter problems in executing method C 208. One problem occurs when primary thread 202 executing method C 208 writes a value to a memory element (object) after speculative thread 203 reads the same memory element. In this case, speculative thread 203 should have read the value written by primary thread 202, but instead read a previous value. In this case, the system causes a speculative thread 203 to roll back so that speculative thread 203 can read the value written by primary thread 202.

Note that the term “memory element” generally refers to any unit of memory that can be accessed by a computer program. For example, the term “memory element” may refer to a bit, a byte or a word memory, as well as a data structure or an object defined within an object-oriented programming system.

FIG. 3 illustrates the state of the system stack during speculative execution of a method in accordance with an embodiment of the present invention. Note that since programming languages such as the Java programming language do not allow a method to modify the stack frame of another method, the system stack
will generally be the same before method B 206 is executed as it is before method C 208 is executed. (This is not quite true if method B 206 returns a parameter through the system stack. However, return parameters are can be explicitly dealt with as is described below.) Referring the FIG. 3, stack 300 contains method A frame 302 while method A 204 is executing. When method A 204 returns, method B 206 commences and method A frame 302 is replaced by method B frame 304. Finally, when method B 206 returns, method C 208 commences and method B frame 304 is replaced by method C frame 306. Note that since stack 300 is the same immediately before method B 206 executed as it is immediately before method C 208 is executed, it is possible to execute method C 208 using a speculative version of stack 300 without first executing method B 206.

Data Structures to Support Speculative Execution

FIG. 4 illustrates how memory is partitioned between stack and heap in accordance with an embodiment of the present invention. In FIG. 4, memory 400 is divided into a number of regions including heap 402, stacks for threads 404 and speculative heap 406. Heap 402 comprises a region of memory from which objects are allocated. Heap 402 is further divided into younger generation region 408 and older generation region 410 for garbage collection purposes. For performance reasons, garbage collectors typically treat younger generation objects differently from older generation objects. Stack for threads 404 comprises a region of memory from which stacks for various threads are allocated. Speculative heap 406 comprises a region of memory from which speculative objects are allocated. Note that speculative heap 406 contains speculative versions of objects that technically have not yet been created. For garbage collection purposes, these speculative versions of objects can be treated as belonging to a generation that is younger than objects within younger generation region 408.
FIG. 5 illustrates the structure of a primary version of object 500 and a speculative version of object 510 in accordance with an embodiment of the present invention.

Primary version of object 500 is referenced by object reference pointer 501. Like any object defined within an object-oriented programming system, primary version of object 500 includes data region 508, which includes one or more fields containing data associated with primary version of object 500. Primary version of object 500 also includes method vector table pointer 506. Method vector table pointer 506 points to a table containing vectors that point to the methods that can be invoked on primary version of object 500.

Primary version of object 500 also includes speculative copy pointer 502, which points to speculative version of object 510, if such speculative version exists. Note that in the illustrated embodiment of the present invention, speculative version 510 is always referenced indirectly through speculative copy pointer 502. Primary version of object 500 additionally includes status word 504, which contains status information specifying which fields from data region 508 have been written to or read by speculative thread 203.

Speculative version of object 510 has the same structure as primary version of object 500. More specifically, speculative version of object 510 includes data region 518, method vector table pointer 516, speculative copy pointer 512 and status word 514. Note that speculative copy pointer 512 points self-referentially back to speculative version of object 510. In a system supporting multiple speculative threads, speculative copy pointer 512 can point to yet another speculative copy of the object associated with yet another speculative thread.

FIG. 6 illustrates the structure of status word 504 in accordance with an embodiment of the present invention. In this embodiment, status word 504 includes checkpoint number 602 and speculative bits 603. Speculative bits 603 includes read bits 604 and write bits 606. Checkpoint number 602 keeps track of
the last time a checkpoint occurred, which typically indicates that a join occurred between primary version of object 500 and speculative version of object 510.

Read bits 604 keep track of which fields within data region 508 have been read since the last checkpoint. Correspondingly, write bits 606 keep track of which fields within data region 508 have been written since the last checkpoint. In one embodiment of the present invention, read bits 604 includes one bit for each field within data region 508. In another embodiment, read bits includes fewer bits than the number of fields within data region 508. In this embodiment, each bit within read bits 604 corresponds to more than one field in data region 508. For example, if there are eight read bits, each bit corresponds to every eighth field. Write bits 606 similarly can correspond to one or multiple fields within data region 508.

Speculative Update Process

Speculative updates occur during selected memory references. For local variable and operand accesses to the system stack, no speculative versions exist and nothing special happens. During read operations by primary thread 202 to objects in the heap 402, again nothing special happens.

Special operations are involved in write operations by primary thread 202 as well as read and write operations by speculative thread 203. These special operations are described in more detail with reference to FIGs.7, 8 and 9 below.

FIG. 7 is a flow chart illustrating operations involved in a write operation to an object by a primary thread 202 in accordance with an embodiment of the present invention. During the write operation, the system checks status word 504 within primary version of object 500 to determine whether a rollback is required (step 702). A rollback is required if speculative thread 203 previously read the data element. A rollback is also required if speculative thread 203 previously
wrote to the object and the current write operation updates both primary version of object 500 and speculative version of object 510.

If a rollback is required, the system causes speculative thread 203 to perform a rollback operation (step 704). This rollback operation allows primary thread 202 to read from (or write to) the object after primary thread 202 writes to the object.

If no rollback is required, the system writes to the primary version of object 500 and the speculative version 510 if such speculative version exists (step 706).

Note that in the embodiment of the present invention illustrated in FIG. 7 the system performs writes to both primary version 500 and speculative version 510. In an alternative embodiment, the system first checks to determine if speculative thread 203 previously wrote to speculative version 510. If not, the system writes to both primary version 500 and speculative version 510. If so, the system only writes to primary version 500.

FIG. 8 is a flow chart illustrating operations involved in a read operation to an object by speculative thread 203 in accordance with an embodiment of the present invention. During this read operation, the system sets a status bit in status word 504 within primary version of object 500 to indicate that primary version 500 has been read (step 802). Speculative thread 203 then reads speculative version 510, if it exists. Otherwise, speculative thread 203 reads primary version 500.

FIG. 9 is a flow chart illustrating operations involved in a write operation to a memory element by speculative thread 203 in accordance with an embodiment of the present invention. If a speculative version 510 does not exist, the system allocates speculative version 510 from speculative heap 406 (step 902). The system next writes to speculative version 510 (step 904). The system also updates status word 504 to indicate that speculative thread 203 has written to the
object if such updating is necessary. Such updating is necessary if primary thread
202 must subsequently choose between writing to both primary version 500 and
speculative version 510, or writing only to primary version 500 as is described
above with reference to FIG. 7.

FIG. 10 is a flow chart illustrating operations involved in a join operation
between primary thread 202 and a speculative thread 203 in accordance with an
embodiment of the present invention. A join operation occurs for example when
primary thread 202 reaches a point in the program where speculative thread 203
began executing. The join operation causes state associated with the speculative
thread 203 to be merged with state associated with the primary thread 202. This
involves copying and/or merging the stack of speculative thread 203 into the stack
of primary thread 202 (step 1002). It also involves merging speculative and
primary versions of objects (step 1004) as well as garbage collecting speculative
heap (step 1006). In one embodiment of the present invention, one of threads 202
or 203 performs steps 1002 and 1006, while the other thread performs step 1004.

FIG. 11 is a flow chart illustrating operations involved in a join operation
between primary thread 202 and a speculative thread 203 in accordance with
another embodiment of the present invention. In this embodiment, speculative
thread 203 carries on as a pseudo-primary thread. As a pseudo-primary thread,
speculative thread 203 uses indirection to reference speculative versions of
objects, but does not mark objects or create versions. While speculative thread
203 is acting as a pseudo-primary thread, primary thread 202 updates primary
versions of objects.

The foregoing descriptions of embodiments of the invention have been
presented for purposes of illustration and description only. They are not intended
to be exhaustive or to limit the invention to the forms disclosed. Accordingly,
many modifications and variations will be apparent to practitioners skilled in the
art. Additionally, the above disclosure is not intended to limit the invention. The scope of the invention is defined by the appended claims.
What Is Claimed Is:

1. A method for updating information associated with a memory primary thread that executes program instructions and a speculative thread that speculatively executes program instructions in advance of the primary thread, the primary thread accessing a primary version of the memory element and the speculative thread accessing a speculative version of the memory element, the method comprising:

   initiating a memory access to the memory element; and
   if the memory access is a write operation by the primary thread,
      performing the write operation to the primary version of the memory element,
      checking status information associated with the memory element to determine if the memory element has been read by the speculative thread,
      if the memory element has been read by the speculative thread, causing the speculative thread to roll back so that the speculative thread can read a result of the write operation, and
      if the memory element has not been read by the speculative thread, performing the write operation to the speculative version of the memory element if the speculative version exists.

2. The method of claim 1, further comprising if the memory access is a read operation by the speculative thread:
   determining if the speculative version of the memory element exists;
   if the speculative version of the memory element exists, reading the speculative version of the memory element;
if the speculative version of the memory element does not exist, reading
the primary version of the memory element; and
updating status information associated with the memory element to
indicate the memory element has been read by the speculative thread.

3. The method of claim 2, further comprising if the memory access is
a write operation by the speculative thread:
determining if the speculative version of the memory element exists;
if the speculative version of the memory element does not exist, creating
the speculative version of the memory element; and
performing the write operation to the speculative version of the memory
element.

4. The method of claim 1, wherein the memory element includes an
object defined within an object-oriented programming system.

5. The method of claim 1, wherein checking status information
associated with the memory element includes determining if the memory element
has been read by a second speculative thread.

6. The method of claim 1, wherein accesses to the speculative version
of the memory element are made indirectly through a pointer associated with the
primary version of the memory element.

7. The method of claim 1, wherein the act of performing the write
operation to the speculative version of the memory element further comprises:
checking status information associated with the memory element to
determine if the speculative version of the memory element has been written by
the speculative thread; and
writing to the speculative version of the memory element only if the
speculative thread has not written to the speculative version of the memory
element.

8. The method of claim 1, wherein if the memory access is a write
operation by the primary thread, the method further comprises:
performing the write operation to the speculative version of the memory
element;
checking status information associated with the memory element to
determine if the speculative version of the memory element has been written by
the speculative thread; and
if the speculative version of the memory element has been written by the
speculative thread, causing the speculative thread to roll back so that the
speculative thread can perform the write operation again in order to undo the write
operation to the speculative version of the memory element by the primary thread.

9. The method of claim 1, further comprising performing a join
operation between the primary thread and the speculative thread when the primary
thread reaches a point in the program where the speculative thread began
executing, the join operation causing state associated with the speculative thread
to be merged with state associated with the primary thread.

10. The method of claim 9, wherein performing the join operation
includes merging the speculative version of the memory element into the primary
version of the memory element and discarding the speculative version of the memory element.

11. The method of claim 9, wherein performing the join operation includes causing the speculative thread to continue executing as a new primary thread and causing the primary thread to continue executing as a new speculative thread.

12. A method for updating information associated with a memory element in a system that supports speculative execution, the system having a primary thread that executes program instructions and a speculative thread that speculatively executes program instructions in advance of the primary thread, the primary thread accessing a primary version of the memory element and the speculative thread accessing a speculative version of the memory element, wherein the memory element includes an object defined within an object-oriented programming system, the method comprising:
   initiating a memory access to the memory element; and
   if the memory access is a write operation by the primary thread,
   performing the write operation to the primary version of the memory element,
   checking status information associated with the memory element to determine if the memory element has been read by the speculative thread,
   if the memory element has been read by the speculative thread, causing the speculative thread to roll back so that the speculative thread can read a result of the write operation, and
if the memory element has not been read by the speculative thread, performing the write operation to the speculative version of the memory element if the speculative version exists;

if the memory access is a read operation by the speculative thread, determining if the speculative version of the memory element exists,

if the speculative version of the memory element exists, reading the speculative version of the memory element,

if the speculative version of the memory element does not exist, reading the primary version of the memory element, and updating status information associated with the memory element to indicate the memory element has been read by the speculative thread;

if the memory access is a write operation by the speculative thread, determining if the speculative version of the memory element exists,

if the speculative version of the memory element does not exist, creating the speculative version of the memory element, and performing the write operation to the speculative version of the memory element; and

performing a join operation between the primary thread and the speculative thread when the primary thread reaches a point in a program where the speculative thread began executing, the join operation causing state associated with the speculative thread to be merged with state associated with the primary thread;
wherein accesses to the speculative version of the memory element are made indirectly through a pointer associated with the primary version of the memory element.

13. An apparatus that supports speculative execution of a program, comprising:
   a primary thread that executes program instructions and accesses a primary version of a memory element;
   a speculative thread that speculatively executes program instructions in advance of the primary thread and that accesses a speculative version of the memory element;
   a mechanism that performs write operations for the primary thread, the mechanism being configured to,
   perform a write operation to the primary version of the memory element,
   check status information associated with the memory element to determine if the memory element has been read by the speculative thread,
   cause the speculative thread to roll back so that the speculative thread can read a result of the write operation if the memory element has been read by the speculative thread, and
   perform the write operation to the speculative version of the memory element if the speculative version exists and if the memory element has not been read by the speculative thread.

14. The apparatus of claim 13, further comprising a mechanism that performs read operations for the speculative thread that is configured to:
    determine if the speculative version of the memory element exists;
read the speculative version of the memory element if the speculative
version of the memory element exists;
read the primary version of the memory element if the speculative version
of the memory element does not exist; and
update status information associated with the memory element to indicate
the memory element has been read by the speculative thread.

15. The apparatus of claim 14, further comprising a mechanism that
performs write operations for the speculative thread that is configured to:
determine if the speculative version of the memory element exists;
create the speculative version of the memory element if the speculative
version of the memory element does not exist; and
perform the write operation to the speculative version of the memory
element.

16. The apparatus of claim 13, wherein the memory element includes
an object defined within an object-oriented programming system.

17. The apparatus of claim 13, further comprising a second speculative
thread that accesses a second speculative version of the memory element.

18. The apparatus of claim 13, wherein the apparatus is configured to
access the speculative version of the memory element indirectly through a pointer
associated with the primary version of the memory element.

19. The apparatus of claim 13, wherein the mechanism that performs
write operations for the speculative thread is further configured to:
check status information associated with the memory element to determine
if the speculative version of the memory element has been written by the
speculative thread; and
write to the speculative version of the memory element only if the
speculative thread has not written to the speculative version of the memory
element.

20. The apparatus of claim 13, wherein the mechanism that performs
write operations for the primary thread is configured to:
   perform the write operation to the speculative version of the memory
element;
   check status information associated with the memory element to determine
if the memory element has been written by the speculative thread; and
   cause the speculative thread to roll back so that the speculative thread can
perform the write operation again if the memory element has been written by the
speculative thread in order to undo the write operation to the speculative version
of the memory element by the primary thread.

21. The apparatus of claim 13, further comprising a mechanism to
perform a join operation between the primary thread and the speculative thread
when the primary thread reaches a point in the program where the speculative
thread began executing, the join operation causing state associated with the
speculative thread to be merged with state associated with the primary thread.

22. The apparatus of claim 21, wherein the mechanism to perform the
join operation is configured to merge the speculative version of the memory
element into the primary version of the memory element and to discard the
speculative version of the memory element.
23. The apparatus of claim 21, wherein the mechanism to perform the join operation is configured to cause the speculative thread to continue executing as a new primary thread and to cause the primary thread to continue executing as a new speculative thread.

24. A computer readable storage medium containing a data structure that supports speculative execution using a primary thread that executes program instructions and a speculative thread that speculatively executes program instructions in advance of the primary thread, the data structure comprising:

   a data storage area for storing data to be operated on by the program instructions;

   a speculative version pointer that points to a speculative version of the data structure that is operated on by the speculative thread; and

   status information that indicates whether the speculative version of the data storage area has been written to or read by the speculative thread.

25. The computer readable storage medium containing a data structure of claim 24, wherein the data structure is an object defined within an object-oriented programming system, and further comprising a method vector table pointer that facilitates references to methods that access the data storage area.

26. The computer readable storage medium containing a data structure of claim 24, wherein the speculative version pointer points self-referentially back to the data structure if the data structure is a speculative version of the data structure.
FIG. 2A
(PRIOR ART)

FIG. 2B

FIG. 3
START 700
WRITE TO PRIMARY VERSION AND SPECULATIVE VERSION (IF IT EXISTS) 702
CHECK STATUS WORD FOR POSSIBLE ROLL BACK 704
IF ROLL BACK, CAUSE SPECULATIVE THREAD TO ROLL BACK AND WRITE TO (OR READ FROM) OBJECT AFTER PRIMARY THREAD WRITES TO OBJECT 706
END 708

FIG. 7

START 800
SET STATUS BIT TO INDICATE MEMORY ELEMENT HAS BEEN READ 802
READ PRIMARY OR SPECULATIVE VERSION OF MEMORY ELEMENT 804
END 806

FIG. 8

START 900
IF SPECULATIVE VERSION DOES NOT EXIST, CREATE SPECULATIVE VERSION 902
UPDATE STATUS WORD IF NECESSARY 903
WRITE TO SPECULATIVE VERSION 904
END 906

FIG. 9
START 1000

COPY/MERGE STACK OF SPECULATIVE THREAD 1002

MERGE OBJECT VERSIONS 1004

GARBAGE COLLECT THE SPECULATIVE HEAP 1006

END 1008

FIG. 10

START 1100

SWITCH SPECULATIVE THREAD TO BE PRIMARY THREAD 1102

END 1104

FIG. 11
### INTERNATIONAL SEARCH REPORT

**International Application No.**
PCT/US 00/12499

#### A. CLASSIFICATION OF SUBJECT MATTER

**IPC 7** G06F 9/38

According to international Patent Classification (IPC) or to both national classification and IPC

#### B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

**IPC 7** G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic database consulted during the international search (name of database and, where practical, search terms used)

**EPO-Internal**

#### C. DOCUMENTS CONSIDERED TO BE RELEVANT

<table>
<thead>
<tr>
<th>Category *</th>
<th>Citation of document, with indication, where appropriate, of the relevant passages</th>
<th>Relevant to claim No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>EP 0 725 334 A (IBM) 7 August 1996 (1996-08-07)</td>
<td>1,9,10, 12,13, 21,22,24</td>
</tr>
<tr>
<td></td>
<td>abstract page 5, line 32 - line 48</td>
<td></td>
</tr>
<tr>
<td></td>
<td>page 6, line 33 - line 42</td>
<td></td>
</tr>
<tr>
<td></td>
<td>page 7, line 49 - line 53</td>
<td></td>
</tr>
<tr>
<td></td>
<td>page 8, line 51 - page 9, line 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>page 10, line 53 - page 11, line 8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>page 11, line 45 - line 54</td>
<td></td>
</tr>
<tr>
<td></td>
<td>page 12, line 21 - line 39</td>
<td></td>
</tr>
<tr>
<td></td>
<td>page 13, line 6 - line 15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>column 3, line 23 - line 54</td>
<td></td>
</tr>
</tbody>
</table>

X Further documents are listed in the continuation of box C.

X Patent family members are listed in annex.

* Special categories of cited documents:

**A** document defining the general state of the art which is not considered to be of particular relevance

**E** earlier document but published on or after the international filing date

**L** document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

**O** document referring to an oral disclosure, use, exhibition or other means

**P** document published prior to the international filing date but later than the priority date claimed

**T** later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

**X** document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

**Y** document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

**Z** document member of the same patent family

Date of the actual completion of the international search

8 September 2000

Date of mailing of the international search report

15/09/2000

Name and mailing address of the ISA

European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk
Tel: (+31-70) 340-2040, Tx: 31 651 epc nl,
Fax: (+31-70) 340-3016

Authorized officer

Moraiti, M
<table>
<thead>
<tr>
<th>Category</th>
<th>Citation of document, with indication, where appropriate, of the relevant passages</th>
<th>Relevant to claim No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>P,A</td>
<td>WO 99 31580 A (AKKARY HAITHAM; CHOW KINGSUM (US); INTEL CORP (US)) 24 June 1999 (1999-06-24) page 22, line 5 - page 23, line 9</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>US 5 872 990 A (WILLIS JOHN CHRISTOPHER ET AL) 16 February 1999 (1999-02-16) column 2, line 40 - column 3, line 41</td>
<td></td>
</tr>
<tr>
<td>Patent document cited in search report</td>
<td>Publication date</td>
<td>Patent family member(s)</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-----------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JP 8249183 A</td>
</tr>
<tr>
<td>US 5961639 A</td>
<td>05-10-1999</td>
<td>NONE</td>
</tr>
<tr>
<td>WO 9931580 A</td>
<td>24-06-1999</td>
<td>AU 1913799 A</td>
</tr>
<tr>
<td>US 5872990 A</td>
<td>16-02-1999</td>
<td>NONE</td>
</tr>
<tr>
<td>GB 2321544 A</td>
<td>29-07-1998</td>
<td>NONE</td>
</tr>
</tbody>
</table>